09/813576

Cof C

PATENT
Attorney Docket No. 43997

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent No. 7,164,676

Confirmation No. 2729

Issued: January 16, 2007

Name of Patentee: Chakraborty

Patent Title: METHOD AND APPARATUS FOR A COMBINED BULK AND TRANSACTIONAL DATABASE SYNCHRONOUS SCHEME

CERTIFICATE OF MAILING

I hereby certify that this paper is being deposited with the United States Postal Service on the date shown with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231, on **July 31, 2007.**

[illegible]

Kirk D. Williams, Esq. 7-31-2007

REQUEST FOR CERTIFICATE OF CORRECTION OF PATENT FOR PATENT OFFICE MISTAKE (37 C.F.R. § 1.322)

Attn: Certificate of Correction Branch
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

It is requested that a Certificate of Correction be issued to correct Office mistakes found the above-identified patent. Attached hereto is a Certificate of Correction which indicates the requested correction. For your convenience, also attached are copies of selected pages (a) from the issued patent with errors highlighted, and (b) from the original application as filed March 21, 2001 with the correct text/instructions.

It appears that page 12 of the application as originally filed was printed out of order, after page 10, in the issued patent. In other words, the pages of the original application were printed in the issued patent in the order of original pages: 1-10, 12, 11, 13-14.

It is believed that there is no charge for this request because applicant or applicants were not responsible for such error, as will be apparent upon a comparison of the issued patent with the application as filed or amended. However, the Assistant Commissioner is hereby authorized to charge any fee that may be required to Deposit Account No. 501430.

Respectfully submitted,
The Law Office of Kirk D. Williams

Date: July 31, 2007

By [illegible] 7-31-2007

Kirk D. Williams, Reg. No. 42,229
One of the Attorneys for Applicants
CUSTOMER NUMBER 26327
The Law Office of Kirk D. Williams
P.O. Box 61538, Denver, CO 80206-8538
303-282-0151 (telephone), 303-778-0748 (facsimile)

AUG 8 2007

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO.	:	7,164,676
DATED	:	January 16, 2007
INVENTOR(S)	:	Chakraborty

It is certified that error(s) appear in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 8, lines 46-67 through Column 9, lines 1-22, text beginning "bulk update)," through text ending "if the transaction" should be inserted in Column 9, line 65 between "a" and "request"

MAILING ADDRESS OF SENDER:
Kirk D. Williams, Reg. No. 42,229
Customer No. 26327
The Law Office of Kirk D. Williams
P.O. Box 61538, Denver, CO 80206

PATENT NO. 7,164,676
No. of additional copies

⇨ NONE (0)

AUG 8 2007

devices. Storage devices **120** typically store computer-executable instructions to be executed by processor **110** and/or data which is manipulated by processor **110** for implementing functionality in accordance with the invention.

As used herein, computer-readable medium is an extensible term including other memory and other storage devices and/or mechanisms.

Standby controller **155** includes processor and/or control logic **160** (hereinafter "processor"), memory **165**, storage devices **170**, switch interface **175**, and one or more internal communications mechanisms **162** (shown as a bus for illustrative purposes). In other embodiments, standby controller **155** may include custom components such as application-specific integrated circuits ("ASICs") to supplement or replace some or all of components **160–175**. In one embodiment, processor **160** controls the operations of standby controller **155** according to the invention. Memory **165** is one type of computer-readable medium, and typically comprises random access memory (RAM), read only memory (ROM), integrated circuits, and/or other memory components. Memory **165** typically stores computer-executable instructions to be executed by processor **160** and/or data which is manipulated by processor **160** for implementing functionality in accordance with the invention. Storage devices **170** are another type of computer-readable medium, and typically comprise disk drives, diskettes, networked services, tape drives, and other storage devices. Storage devices **170** typically store computer-executable instructions to be executed by processor **160** and/or data which is manipulated by processor **160** for implementing functionality in accordance with the invention.

Switch interfaces **125** and **175** allow bulk and transactional communication between active controller **105** and standby controller **155** (and with switching elements **140**) over communications link **141**. This communication typically includes packets or control signals for synchronizing an active connection database (and/or other data structures) maintained by active controller **105** in memory **115** and/or storage devices **120** with a secondary database (and/or other data structures) maintained by standby controller **155** in memory **165** and/or storage devices **170**.

As indicated by FIGS. 1A–B, synchronization methods and apparatus according to the invention may be used in an unbounded number of configurations and systems. FIG. 1B illustrates a block diagram of another embodiment of the invention for synchronizing two databases using a bulk and transactional database synchronization scheme. Active element with first database **190** and standby element with second database **194** communicate bulk and transactional messages or signals over communications network **192** and links **191** and **193**. In some embodiments, active element with first database **190** and standby element with second database **194** are located in separate components, devices or systems. In some embodiments, active element with first database **190** and standby element with second database **194** are implemented as part of a single computer or communications device, with communications network **192** being, for example, but not limited to a data bus or some other internal information sharing mechanism such as message passing or shared memory.

FIG. 2 illustrates one data structure **200** used in one embodiment of a bulk and transactional database synchronization scheme according to the invention. In one embodiment, the primary database is divided into synchronization groups **210** of zero or more entries or transactions (hereafter "entries") **213**. Entries **213** may each contain a plurality of data elements to be synchronized, which may be maintained

in any data structure, such as a linked list, array, set, load balanced tree, etc. In one embodiment, the particular entries within a plurality of entries **213** are maintained in a sorted order. In one embodiment, each of the entries **213** contain a dirty flag (e.g., a bit) which may be used to indicate whether a particular entry **213** requires synchronization or has been synchronized with another database. Groups **210** may be divided in most any way, such as that using a hashing function or based on a value of a location, address, connection identifier, etc.

Initially, all the groups **210** are marked as requiring bulk synchronization. Systematically, multiple entries **213** from one or more groups **210** are combined into a bulk update message and relayed to a secondary device or component to bulk update the secondary database. In one embodiment, data structure **200** includes a dirty flag field **211** to indicate whether a particular group **210** is subject to a bulk update technique. In one embodiment, data structure **200** includes a data field **212** which may reference or include entries **213** (e.g., connections, other data) of the actual data being synchronized within each particular group **210** of the multiple groups **210**. In one embodiment, each entry **213** includes an entry dirty flag to indicate whether or not it is subject to a bulk update.

FIG. 3A illustrates one embodiment of a bulk update message **300**. A bulk update message refers to any mechanism which can be used to communicate multiple entries between two databases or data structures. Typically, bulk update messages are used to systematically communicate groups of entries contained in a first database or data structure, rather than to communicate entries in real-time as they are received. In one embodiment, bulk update message bulk update message sent, for example, from an active controller to a standby controller of a switching system or router. Message format **300** includes a header field **301**, transaction acknowledgement request field **302** typically used to request an acknowledgement from the standby controller, entry fields **304–306** corresponding to the data being synchronized, and a data group identifier field **303** to indicate to which group entries **304–306** belong.

FIG. 3B illustrates one embodiment of a bulk update acknowledgement message **310** sent, for example, from a standby controller to an active controller to acknowledge receipt of a bulk update message **300** (FIG. 3A) and/or updating of its data structure. As shown, bulk update acknowledgement message **310** includes a header field bulk update), then processing proceeds to process block **410** where a set of entries are retrieved from one of the dirty groups and the dirty indicators for the selected entries are reset. In one embodiment, entries within a set of entries are maintained in a sorted order or using another data structure to make the step of getting more entries more efficient. Next, if all remaining entries to be bulk updated from the particular group have not been selected as determined in process block **412**, then the entries are put into a bulk update message and the message being sent to the standby component in process block **414**, and processing returns to process block **406** to process more dirty entries and groups. Otherwise, in process block **416**, the transaction acknowledgement field of the bulk update message for the selected entries is set in process block **416**, and the bulk update message is sent to the standby component in process block **418**. Next, in process block **420**, the process waits until it receives a transaction acknowledgement message from the standby component. In another embodiment, a separate process thread is used to receive the transaction acknowledgement messages. If, as determined in process block **422**, there remains a dirty element in the group for which the transaction acknowledgement message was received in process block **420**, processing returns to process block **406** to process more dirty entries and groups. Otherwise, the group dirty flag is reset for the group for which the transaction acknowledgement message was received in process block **420** and processing returns to process block **406** to check to see if there are more dirty groups to process. When there are no more dirty groups as determined in process block **406**, then the bulk updating is finished as indicated by process block **408**.

FIG. **4B** illustrates one embodiment of a process of an active component for receiving and handling a new transaction (e.g., a connection update or other primary database update, etc.). Processing begins at process block **430**, and proceeds to process block **432** wherein the transaction request is received from another element, component, system, etc. The local database of the active controller is updated with the new transaction request in process block **434**. In one embodiment, new entries and possibly new groups may be created and added to the local database. Next, if the transaction **311**, a transaction acknowledgement flag field **312** to indicate the acknowledgment, and a data group identifier field **313** to indicate to which group the acknowledgement refers. In one embodiment, a bulk update acknowledgement message **310** is used to inform the active controller that the standby controller has acted upon all entries in a group sent so far, thus eliminating ordering issues and providing an orderly transition of the active controller from a bulk update mode to a transactional update mode for the group or groups specified in data group identifier field **313**.

FIG. **3C** illustrates one embodiment of a transactional update message **320** sent, for example, from the path manager to an active controller or from an active controller to a standby controller. A transactional update message refers to any mechanism which can be used to communicate typically one, but also multiple entries in real-time or almost real-time, and are typically used during a transactional synchronization mode. For example, shortly after updating its database with an transaction, an active controller may send to a standby controller the transaction using a transactional update message **320**. In one embodiment, transactional update message **320** includes a header field **321** and a transactional update entry field **322**. In one embodiment, a transactional update message contains either one or more than one transactional update entry with the entries of a message being from the same or different groups.

The operation of one embodiment of a system using a combined bulk and transactional database synchronization scheme is further illustrated by the flow diagrams of FIGS. **4A–B** and **5**. The operations illustrated in FIGS. **4A–B** and **5** typically operate in parallel, and may be embodied by numerous implementation including being performed by separate hardware threads or separate threads of one or more processes.

FIG. **4A** illustrates one embodiment of a process of an active component bulk updating a standby component. Process begins at process block **400**, and proceeds to process block **402** where a standby initialization event is received or recognized to indicate that a bulk update is required. Next, in process block **404**, all database entries and groups are marked as dirty to indicate that they are subject to a bulk update process. Next, as determined in process block **406**, while groups remain dirty (i.e., are subject to a request belongs to a new bulk update group as determined in process block **436**, then the standby database will be transactional

INSERT HIGHLIGHTED TEXT HERE

updated in process block **442** with the new transaction request (rather than as part of a bulk update). Otherwise, if the group to which the new transaction request is not dirty as determined in process block **438**, the standby database will be transactional updated in process block **442** with the new transaction request. Otherwise, the entry previously added to the local database in process block **432** (and corresponding to the received transaction request) is marked as dirty in process block **440** (and the standby database will be bulk updated with this entry). Processing returns to process block **432** to receive and process more transactions.

FIG. **5** illustrates one embodiment of a process of a standby component for receiving bulk and transactional updates from an active component and updating its database. Processing begins at process block **500**, and proceeds to process block **502** wherein an update message is received from an active component. Next, if the message corresponds to a bulk update as determined in process block **504**, then the local database of the standby component is updated with the entries included in the received bulk update message in process block **506**. If an acknowledgement was requested in the received bulk update message as determined in process block **508**, then a transaction acknowledgment message is sent to the active component in process block **510**. Otherwise, if the message corresponds to a transactional update as determined in process block **504**, then the local database of the standby component is updated with the entry or entries included in the received transactional update message in process block **512**. Processing returns to process block **502** to receive and process more update messages.

In view of the many possible embodiments to which the principles of our invention may be applied, it will be appreciated that the embodiments and aspects thereof described herein with respect to the drawings/figures are only illustrative and should not be taken as limiting the scope of the invention. For example and as would be apparent to one skilled in the art, many of the process block operations can be re-ordered to be performed before, after, or substantially concurrent with other operations. Also, many different forms of data structures could be used in various embodiments. The invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.

What is claimed is:

1. A method for duplicating a plurality of entries of a first database to a second database and maintaining said entries in synchronization between the first database and second database using a combined bulk and transactional update scheme, wherein bulk updating refers to the initial updating of the second database with sets of multiple entries from the first database and transactional updating refers to the updating of entries as they are applied to the first database to the second database for entries that are no longer subject to said initial bulk updating, wherein said updating of the second database includes committing the entry or entries to the second database; the method comprising:

initializing each of a plurality of groups of said entries as requiring bulk updating prior to commencing updating of the second database with said entries, wherein, at least one of said groups of entries includes a plurality of said entries, and each of said entries are included in one of said groups; wherein said groups are associated with indications identifying which groups still require said bulk updating; wherein said initializing includes initializing said indications to identify that all of the groups of entries still require said bulk updating; and after said initializing, systematically bulk updating

AUG 8 2007

From Application Filed March 21, 2001

indicate whether a particular entry 213 requires synchronization or has been synchronized with another database. Groups 210 may be divided in most any way, such as that using a hashing function or based on a value of a location, address, connection identifier, etc.

Initially, all the groups 210 are marked as requiring bulk synchronization. Systematically, multiple entries 213 from one or more groups 210 are combined into a bulk update message and relayed to a secondary device or component to bulk update the secondary database. In one embodiment, data structure 200 includes a dirty flag field 211 to indicate whether a particular group 210 is subject to a bulk update technique. In one embodiment, data structure 200 includes a data field 212 which may reference or include entries 213 (e.g., connections, other data) of the actual data being synchronized within each particular group 210 of the multiple groups 210. In one embodiment, each entry 213 includes an entry dirty flag to indicate whether or not it is subject to a bulk update.

FIG. 3A illustrates one embodiment of a bulk update message 300. A bulk update message refers to any mechanism which can be used to communicate multiple entries between two databases or data structures. Typically, bulk update messages are used to systematically communicate groups of entries contained in a first database or data structure, rather than to communicate entries in real-time as they are received. In one embodiment, bulk update message bulk update message sent, for example, from an active controller to a standby controller of a switching system or router. Message format 300 includes a header field 301, transaction acknowledgement request field 302 typically used to request an acknowledgement from the standby controller, entry fields 304-306 corresponding to the data being synchronized, and a data group identifier field 303 to indicate to which group entries 304-306 belong.

FIG. 3B illustrates one embodiment of a bulk update acknowledgement message 310 sent, for example, from a standby controller to an active controller to acknowledge receipt of a bulk update message 300 (FIG. 3A) and/or updating of its data structure. As shown, bulk update acknowledgement message 310 includes a header field

AUG 8 2007

From Application filed March 21, 2001

311, a transaction acknowledgement flag field 312 to indicate the acknowledgment, and a data group identifier field 313 to indicate to which group the acknowledgement refers. In one embodiment, a bulk update acknowledgement message 310 is used to inform the active controller that the standby controller has acted upon all entries in a group sent so far, thus eliminating ordering issues and providing an orderly transition of the active controller from a bulk update mode to a transactional update mode for the group or groups specified in data group identifier field 313.

FIG. 3C illustrates one embodiment of a transactional update message 320 sent, for example, from the path manager to an active controller or from an active controller to a standby controller. A transactional update message refers to any mechanism which can be used to communicate typically one, but also multiple entries in real-time or almost real-time, and are typically used during a transactional synchronization mode. For example, shortly after updating its database with an transaction, an active controller may send to a standby controller the transaction using a transactional update message 320. In one embodiment, transactional update message 320 includes a header field 321 and a transactional update entry field 322. In one embodiment, a transactional update message contains either one or more than one transactional update entry with the entries of a message being from the same or different groups.

The operation of one embodiment of a system using a combined bulk and transactional database synchronization scheme is further illustrated by the flow diagrams of FIGs. 4A-B and 5. The operations illustrated in FIGs. 4A-B and 5 typically operate in parallel, and may be embodied by numerous implementation including being performed by separate hardware threads or separate threads of one or more processes.

FIG. 4A illustrates one embodiment of a process of an active component bulk updating a standby component. Process begins at process block 400, and proceeds to process block 402 where a standby initialization event is received or recognized to indicate that a bulk update is required. Next, in process block 404, all database entries and groups are marked as dirty to indicate that they are subject to a bulk update process. Next, as determined in process block 406, while groups remain dirty (i.e., are subject to a

From Application Filed March 21, 2001

bulk update), then processing proceeds to process block 410 where a set of entries are retrieved from one of the dirty groups and the dirty indicators for the selected entries are reset. In one embodiment, entries within a set of entries are maintained in a sorted order or using another data structure to make the step of getting more entries more efficient. Next, if all remaining entries to be bulk updated from the particular group have not been selected as determined in process block 412, then the entries are put into a bulk update message and the message being sent to the standby component in process block 414, and processing returns to process block 406 to process more dirty entries and groups. Otherwise, in process block 416, the transaction acknowledgement field of the bulk update message for the selected entries is set in process block 416, and the bulk update message is sent to the standby component in process block 418. Next, in process block 420, the process waits until it receives a transaction acknowledgement message from the standby component. In another embodiment, a separate process thread is used to receive the transaction acknowledgement messages. If, as determined in process block 422, there remains a dirty element in the group for which the transaction acknowledgement message was received in process block 420, processing returns to process block 406 to process more dirty entries and groups. Otherwise, the group dirty flag is reset for the group for which the transaction acknowledgement message was received in process block 420 and processing returns to process block 406 to check to see if there are more dirty groups to process. When there are no more dirty groups as determined in process block 406, then the bulk updating is finished as indicated by process block 408.

FIG. 4B illustrates one embodiment of a process of an active component for receiving and handling a new transaction (e.g., a connection update or other primary database update, etc.). Processing begins at process block 430, and proceeds to process block 432 wherein the transaction request is received from another element, component, system, etc. The local database of the active controller is updated with the new transaction request in process block 434. In one embodiment, new entries and possibly new groups may be created and added to the local database. Next, if the transaction